Things to integrate with Mantis 1447

Relates to: SV Mantis items 1702, 1447
Applies to: IEEE 1800-2008 draft 4

Version 1, 27 Nov 2007

Mainly trying to ensure assignment compatibility amongst all kinds of non-associative unpacked arrays, provided that their elements are of assignment-compatible types and the sizes are OK.

Jonathan Bromley, 27 November 2007

CHANGE final paragraph of 7.4 as follows:

~~Arrays can be *fixed* or *dynamic*. Fixed-size unpacked arrays can be multidimensional and have fixed storage allocated for all the elements of the array. Each dimension of an unpacked array can be declared as having a fixed or unfixed size. A dynamic array allocates storage for elements at run time along with the option of changing the size of one of its dimensions. An associative array allocates storage for elements individually as they are written. Associative arrays can be indexed using arbitrary data types. A queue type of array grows or shrinks to accommodate the number of elements written to the array at run time.~~

An array may have any number of unpacked dimensions. Each unpacked dimension of an array shall be declared in any one of the following ways:

- as a fixed-size dimension (see **7.4.5**)
- as a dynamic array dimension (see **7.5**)
- as an associative array dimension (see **7.9**)
- as a queue dimension (see **7.11**)

A dynamic array allocates storage for elements at run time along with the option of changing the size of one of its dimensions. An associative array allocates storage for elements individually as they are written. Associative arrays can be indexed using arbitrary data types. A queue type of array grows or shrinks to accommodate the number of elements written to the array at run time.

ADD text in 7.6 as follows:

A dynamic array or queue can be assigned to a fixed-size array ~~of an equivalent~~ having elements of assignment-compatible type if the size of the dynamic array or queue dimension is the same as the length of the fixed-size array dimension. Unlike assigning with a fixed-size array, this operation requires a run-time check that can result in an error, in which case no operation shall be performed.

```
int A[100:1]; // fixed-size array of 100 elements
int B[] = new[100]; // dynamic array of 100 elements
int C[] = new[8]; // dynamic array of 8 elements
A = B; // OK. Compatible type and same size
A = C; // type check error: different sizes
```

A dynamic array, queue or one-dimensional fixed-size array can be assigned to a queue having elements of assignment-compatible type.

A dynamic array, queue or ~~a one-dimensional~~ fixed-size array can be assigned to a dynamic array ~~of a compatible~~ having elements of assignment-compatible type. In this case, the assignment creates a new dynamic array with a size equal to the length of the fixed-size array. For example:

CHANGE 7.7 as follows:

Arrays can be passed as arguments to tasks or functions. The rules that govern array argument passing by value are the same as for array assignment (see 7.6). When an array argument is passed by value, a copy of the array is passed to the called task or function. This is true for all array types: fixed-size, dynamic, queue, or associative.

If a dimension of a formal is unsized (unsized dimensions can occur in dynamic arrays, queues and in formal arguments of import DPI functions), then any size of the corresponding dimension of an actual is accepted.

For example, the declaration

```
task fun(int a[3:1][3:1]);
```

declares task `fun` that takes one argument, a two-dimensional array with each dimension of size 3. A call to `fun` must pass a two-dimensional array and with the same dimension size 3 for all the dimensions. For example, given the above description for `fun`, consider the following actuals:

```
int b[3:1][3:1]; // OK: same type, dimension, and size
int b[1:3][0:2]; // OK: same type, dimension, & size (different ranges)
reg b[3:1][3:1]; // error: incompatible element type
event b[3:1][3:1]; // error: incompatible type
int b[3:1]; // error: incompatible number of dimensions
int b[3:1][4:1]; // error: incompatible size (3 vs. 4)
```

A subroutine that accepts a one-dimensional fixed-size array can also be passed a dynamic array or queue of a compatible type, provided it is of the same size.

For example, the declaration

```
task bar( string arr[4:1] );
```

declares a task that accepts one argument, an array of 4 strings. This task can accept the following actual arguments:

```
string b[4:1]; // OK: same type and size
string b[5:2]; // OK: same type and size (different range)
string b[] = new[4]; // OK: same type and size, requires run-time check
```

A subroutine that accepts a dynamic array or queue can be passed a dynamic array, queue ~~of a compatible type~~ or ~~a~~ one-dimensional fixed-size array of a compatible type.

For example, the declaration

```
task foo( string arr[] );
```

declares a task that accepts one argument, a dynamic array of strings. This task can accept any one-dimensional array ~~of strings~~, queue or ~~any~~ dynamic array of strings.

An import DPI function that accepts a one-dimensional array can be passed a dynamic array of a compatible type and of any size if formal is unsized and of the same size if formal is sized. However, a dynamic array cannot be passed as an argument if the formal is an unsized output.